Language Impact on Vectorization: Vector Programming in Fortran

Florian Wende

Zuse Institute Berlin
Synopsis

SIMD parallelism is one key mechanism to be addressed when writing code for modern multi- and manycore processors.

How SIMD-friendly is my code?
How to adapt and modernize?
How to avoid compiler specific modifications?

Performance?
autovectorizer
vs.
explicit vector programming (compiler directives)
vs.
manual vector programming (assembly, intrinsics)
Vector Programming in Fortran

Language constructs already enabling SIMD vectorization

- Array notation: \( a(:) = 2.0 \times b(:); \) call \( f(s, a(:), b(:)) \)
- \textbf{where} construct for conditional array operations
  ```fortran
  where (a > 0.0)
    a = log(a)
  elsewhere
    a = 0.0
  end where
  ```

- \textbf{forall} construct to define per-element sequence of operations
  ```fortran
  forall (i=1:16)
    a(i) = 2.0 \times b(i)
    a(i) = \exp(a(i))
  end forall
  ```

👍 Clear code: compiler can do SIMD
👎 You do not get the mask!
👎 Nested \texttt{where} seems not so efficient!
More control with vector data types: define them yourself

- Define SIMD width: `#define SIMDWIDTH 8` (e.g. for IMCI, 64-bit words)
- Vectors containing 64-bit floating point elements + vector mask
  
  ```fortran
  type::simd_real8
  real*8::x(0:SIMDWIDTH-1)
  end type simd_real8

  type::simd_mask8
  logical::x(0:SIMDWIDTH-1)
  end type simd_mask8
  ```

- Compile code with `-align_byte` (Intel specific) or use compiler directive `!dir$ attributes align (Intel specific)` in code
Vector Programming in Fortran – Explicit++

How to adapt the code?

```fortran
... do i=1,n
   call foo(x1(i),x2(i),y(i))
enddo...

foo refers to the loop body or a SIMD function

type(simd_real8)::buffer_x1,buffer_x2,buffer_y

type(simd_mask8)::m

... do i=1,n,SIMDWIDTH
!$omp simd
   do ii=0,SIMDWIDTH-1
     if ((i+ii).le.n)
       m%x(ii)=.true.
       buffer_x1%x(ii)=x1(i+ii)
       buffer_x2%x(ii)=x2(i+ii)
     else
       m%x(ii)=.false.
     endif
   enddo
   call foo_simd(buffer_x1,buffer_x2,buffer_y,m)
!$omp simd
   do ii=0,SIMDWIDTH-1
     if (m%x(ii)) y(i+ii)=buffer_y%x(ii)
   enddo
enddo
...

Load data into vector buffers + create mask

SIMD function call

Store content of vector buffers to output array
```
Vector Programming in Fortran – Explicit++

subroutine foo(x1,x2,y)
real*8:: x,y,a
a=x1*x2
if (a<=0.0)
  y=0.0
  return
endif
a=sqrt(a)
y=log(a)
end subroutine foo

subroutine foo_simd(x1,x2,y,m0)
type(simd_real8):: x1,x2,y,a
type(simd_mask8):: m0,m1
integer:: i
logical:: true_for_any=.false.
!$omp simd reduction(.or.:true_for_any)
do i=0,SIMDWIDTH-1
  a%x(i)=x1%x(i)*x2%x(i)
  if (m0%x(i).and.a%x(i)>0.0) then
    m1%x(i)=.true.
    true_for_any=.true.
  else
    m1%x(i)=.false.
    y%x(i)=0.0
  endif
endo
if (.not.true_for_any) return
!$omp simd
do i=0,SIMDWIDTH-1
  if (m1%x(i)) then
    a%x(i)=sqrt(a%x(i))
    y%x(i)=log(a%x(i))
  endif
endo
end subroutine foo_simd

Similar for
- conditional code blocks
- while loops on SIMD lanes
- nested SIMD-enabled functions

Mask m0 from caller

Early return

Return if all SIMD lanes are inactive
Vector Programming in Fortran – Explicit++

Performance of early return on a Haswell System and Xeon Phi:
- random input: \( x1() \) and \( x2() \) from \([-100.0,+100.0]\)
- only half the number of SIMD lanes active on average

<table>
<thead>
<tr>
<th></th>
<th>Haswell, AVX2*</th>
<th>Xeon Phi, IMCI*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>150ms</td>
<td>1310ms</td>
</tr>
<tr>
<td>Explicit vectorization</td>
<td>136ms</td>
<td>985ms</td>
</tr>
<tr>
<td>Explicit++ vectorization</td>
<td>81ms</td>
<td>575ms</td>
</tr>
<tr>
<td>SIMD intrinsics (C)</td>
<td>67ms</td>
<td>395ms</td>
</tr>
</tbody>
</table>

*Just one thread used for execution on both CPU and Xeon Phi*
We use explicit(++) vectorization in Fortran to vectorize along the calling tree.

Xeon Phi, 8 concurrent offloads

- Explicit++ Vectorization (28 Threads)
- Explicit Vectorization (28 Threads)
- Explicit++ Vectorization (7 Threads)
- Explicit Vectorization (7 Threads)
- Original (CPU)

Execution time [s]

- ggaall_grid
- ggaall
  |  +-calc_exchwpbe_sp
  |  +-ex_sr
  |  +-vr_sr
  |  +-wpbe_spline
  |  +-wpbe_splin2
  |  +-corunsp Kobe
Thank You
Vector Programming in Fortran – Explicit++

Performance of early return on a Haswell System and Xeon Phi:

- random input: x1() and x2() from [-100.0,+100.0]
- only half the number of SIMD lanes active on average

<table>
<thead>
<tr>
<th></th>
<th>Haswell, AVX2*</th>
<th>Xeon Phi, IMCI*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>150ms</td>
<td>1310ms</td>
</tr>
<tr>
<td>Explicit vectorization</td>
<td>136ms</td>
<td>985ms</td>
</tr>
<tr>
<td>Explicit++ vectorization</td>
<td>81ms</td>
<td>575ms</td>
</tr>
<tr>
<td>SIMD intrinsics (C)</td>
<td>67ms</td>
<td>395ms</td>
</tr>
<tr>
<td>SIMD intrinsics (C call from Fortran, entire function)</td>
<td>73ms</td>
<td>-</td>
</tr>
<tr>
<td>SIMD intrinsics (C call from Fortran, instruction level)</td>
<td>90ms</td>
<td>-</td>
</tr>
</tbody>
</table>

*Just one thread used for execution on both CPU and Xeon Phi